International Journal of Information, Control and Computer Sciences
ISSN: 2517-9942
Vol:11, No:6, 2017

Improving Security by Using Secure Servers Communicating via Internet with Standalone Secure Software

Carlos Gonzalez

***Abstract*—**This paper describes the use of the Internet as a feature to enhance the security of our software that is going to be distributed/sold to users potentially all over the world. By placing in a secure server some of the features of the secure software, we increase the security of such software. The communication between the protected software and the secure server is done by a double lock algorithm. This paper also includes an analysis of intruders and describes possible responses to detect threats.

***Keywords*—**Internet, secure software, threats, cryptography process.

I. Introduction

THIS paper describes how we can use the Internet in the design and implementation of secure software. This is done by having in a secure server some of the more sensitive software features of the secure software. This secure software will communicate and execute the sensitive features now located in the secure server via the Internet. This process will provide procedures to increase the security of our standalone software when this software is distributed and used by users all over the world.

When designing secure software, it is difficult to gage and prevent all possible outcomes when the user is in complete control of all your software. The only possible defenses are imbedded into the software leaving this software to its own self-protect features [1], [6]-[8].

Internet servers today have improved their security and provide reasonable security for users. If we assume that we have a server, and that such a server is under our control and is using server protection technology, it is therefore better protected than a software program in the hands and control of a user. Then, using the Internet to host part or parts (the most sensitive, crucial, secret, etc.) of the delivered software is a sensible idea.

II. Requirements

Following is a list of requirements for the use of this proposed technology and methodology:

- **Internet Access:** It is required for the use of this technology and methodology that our secure software be able to have access and make use of the Internet. If this is not the case, this technology will not work.
- **Hard Real-time Requirements**: Knowing that communications over the internet do not guarantee a set response time. The response times may vary over-time. Therefore, any software requiring strict real-time response times will not be a candidate to use the Internet as an internal security feature.

Carlos Gonzalez is with the Universidad Autonoma de Coahuila, Arteaga Mexico (e-mail: gonzalezc757@gmail.com).

III. Methodology

We have several advantages in using the Internet. One such advantage is having most of the critical or sensitive code protected in a secure server under our control [3], [4].

From a design point of view, first it is necessary to decide what we need to send to the secure server.

A. Secure Server Features

Here is a partial list of the features that can be in our secure server, which we will call "Secure Server Features" (SSF) [2], [5]:

- The most sensitive algorithms or procedures.
- The most crucial algorithms or procedures for running of the software.
- The most secret algorithms or procedures of the software
- The most important data (i.e. all or part of a database)

It is understood that our secure software communicates with the secure server using the Internet via secure protocols. These protocols like https and other crypto algorithms will provide security for the communications of the server to/from the secure software.

B. Security Risks

Following is a list of security risk types for the software features that are identified as security risks.

- **Maximum:** These features are very high risk. If a feature is known to any non-authorized user the result could be loss of life, materials, or security integrity.
- **High:** These features, if known by non-authorized users, may cause for example economic losses for the company/ country/individual, or divulge valuable proprietary information.
- **Medium:** For these features, we prefer no access be available to non-authorized users, but if obtained, the situation is not catastrophic.
- **Confidential:** These features utilize algorithms or procedures publicly known, but it is better for the overall security of the system to keep them hidden.

C. Simple Algorithm

A simple algorithm and methodology that can be used is as:

- At the entrance of the secure software, the user is going to be required to be authenticated, and this is done by the secure software sending to the server the user's password (encrypted) and waiting for a response, which will be in the form of a token with information generated by the server that could be for example a combination of the user ID and password (see complete communication algorithm below).
- With the returned token, the secure software will add this token when calling to perform any of the SSF existent in the secure software. For example, an algorithm calculation, when the server receives this request, it checks to see if the sent token (the one returned with the password authentication) is the same as the token currently received. If the tokens are not the same, an "Enemy on Board" (EOB) signal is sent back to the secure software. If the tokens agree, then the algorithm will be executed, and a result is sent back to the secure server. In the following sections of this paper, we will explore some of the actions that the secure server can take having detected an EOB signal.
- With the previous actions, at least we know that no answer from the server will go out un-authenticated.

The decision of which features will be in the secure server (the selection of the SSF) will be made at the time of the software design and development. The designers will have basically three options for each of the selected at risk features:

- Have the features in the server be static. The code in the secure software will be hard coded for calls to the secure server whenever that feature is called (executed). In other words, if a feature like an algorithm is decided to reside in the secure server, then all the call in the secure software of this algorithm will be calls for the remote execution of this algorithm in the remote secure server, which is the only place where the executable code of such algorithm exists.
- Have the features in the server be dynamic. The executable code of the feature resides in the secure software, and a copy of such an executable code will reside in the secure server. The idea here is that if the software detects a situation of risk, then it will immediately erase the executable code for the feature, and replace it with a call to the secure server to execute from now on this feature remotely from the secure server. The advantage of this dynamic mode is that when the secure software is detected to be used by friendlies, then it will not need to interact (spend time) communicating over the Internet. The disadvantage is that we need to be sure of the type of user of our software, and when at risk, have the time to do the erasing and changing of all the features using this mode.
- Have all the security features located in the server. The Maximum and High security type features will never be removed from there. The other features may migrate to the secure software when the secure software feels with high certainty that the user is a friendly user. The first features to migrate will be the confidential features and the medium type last.

In case of any migration from the server to secure software is required, this will be done following cyber-ecological procedures. This concept is explained later on.

We propose that on the first user interaction with the secure software, the secure software should contact the secure server and establish a user ID, maybe set up a cookie. On all other interactions, it uses the user ID to communicate with the secure server. We recommend establishing a procedure to change the user ID (i.e. after every n interaction, at random intervals, etc.). This is done for security purposes to make sure the user was not hacked himself/herself and some other user has access to their ID without their knowledge. All the changes for the ID will be done transparently and without the user's knowledge.

Once the user ID is set, the next question is:

- Is the user type and location well established?
- Yes, the location and user type have been properly established, then:
- If it is a friendly location and a non-threatening user, keep the secure software as it is now.
- If either the location is unfriendly or the user type is a foe, implement all the proper responses to such a threat (i.e. if dynamic features are present, react as described above)? The response should include a signal to the secure server of the responses taken.
- No, either the location has not been properly established or the user type not clearly defined. Keep the system as it is now, and keep taking user and location measurements.

If the Internet usage is part of self-protected software [9], [10], then we will send to the secure server all the information we have about the user and the location of the software. The IP address could give good location information, except in the case of virtual IPs. At the moment, there are no good technical solutions for the virtual IP problem. What we propose is to have a list of the known virtual IP providers, and make the secure software unworkable whenever any of these virtual IPs try to use our secure self-protected software. It is not a complete or clean solution, but it will help in many cases.

D. Communications Process

The software and the Secure Server communicates via a two locks algorithm. This algorithm works as follows:

- When the software needs to communicate with the Secure Server, it first sends an encrypted request to the Secure Server using the Public Key of the Secure Server (PSS)
- The Secure Server decrypts the request using his secret key (SSS). Once this server sees that it is a request to start communication, it generates an ID for the message.
- The Secure server encrypts the generated ID for the software using the software public key (PSO), and sends the message to the software.
- Upon receiving the response from the Secure Server, the software decrypts the message using its secure key (SSO), and saves the value of the ID sent by the Secure Server.

- The software now generates a message which contains the sent ID, and the processing request M encrypted using the secure server key (PSS).
- The Secure Server decrypts the message using the user's secure key (SSO). This message contains an ID, and the request for processing M. If the IDs match (i.e. the one sent and the one received are the same), it proceeds to do the processing of request M.
- Once the processing of M is done, it proceeds to generate an answer back for the software. The message will contain the ID, and the results of the processing N. This information is encrypted using PSO and sent to the software.
- The Software receives the message from the Secure Server and decrypts it using the user's SSO. If the ID coincides, then the software proceeds to manipulate and continue with the response N. This concludes the algorithm cycle.

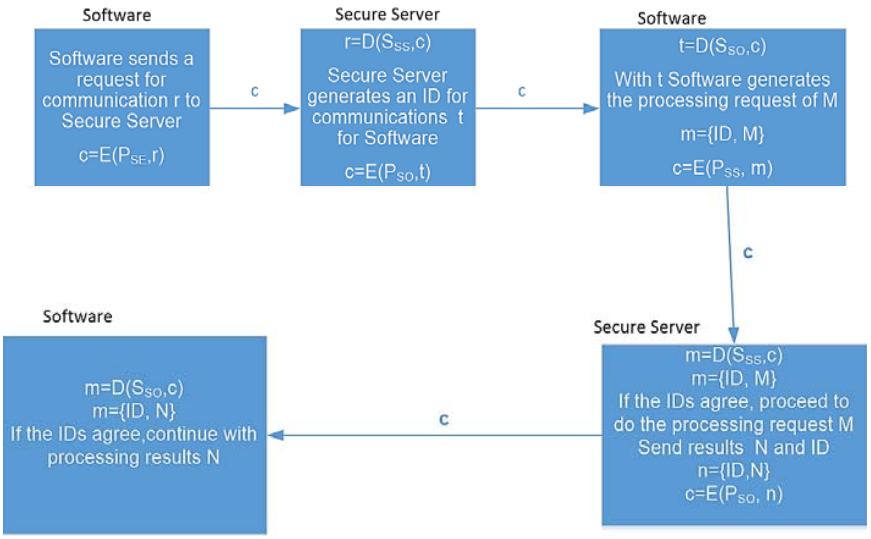

Fig. 1 Software and Secure Server Communication

Fig. 1 shows a diagram of the previously described algorithm.

IV. Intruders

We define four types of intruders:

- Level-1: A **casual attacker.** The attacker has the software and he/she is not technically knowledgeable to retrieve data or algorithms from the machine code software.
- Level-2: A **hacker attack**. This attacker has the knowledge to retrieve data or algorithms from the machine code sources of the software. Attacks of this kind need to have security procedures used for the development of code.
- Level-3: An **institution attack**. This attack is done by an institution with all the resources of such an institution. The most common cases are industrial espionage
- Level-4: A **government attack**. This attack is done by a government agency with all the resources (technical and legal) available for such an agency.

To define the level of user's threat, we have to evaluate all the information available about the user and the current location.

When analyzing the user's threat level, we should keep the following in mind:

- Level-1: A casual attacker. A minimum of security is needed.
- Level-2: A hacker attack. This intruder may or may not have initial plans for economic gains for the intrusion. In most cases, it is the intellectual challenge that motivates this intruder (i.e. hacker), but economic gains may not be very far behind.
- Level-3: An institution attack. The economic gains are the main reason for the intrusion. In most cases with enough time and money, any secure self-protected software may be cracked. Therefore, the developing team should always work with the goal of making the intruder's effort needed to break the secure code high enough for them not to be cost effective.
- Level-4: A government attack. Since in most cases with enough time and money any secure self-protected software may be cracked, it is recommended that techniques for intruder detection [11], [12] as well as the user detection described in this paper with the respective actions to take (covert and not-covert) be included in the secure code This level of protection requires the use of the most sophisticated security algorithms.

V. Threats and Actions

A. Threats

To define the level of user's threat we have to evaluate all the information available about the user and the current location of the device that is running the secure software.

Following is a list of situations that help us determine the level of user threat:

1) If we detect that our software has been modified (i.e. the routine to use the GPS was by-passed), then we know that we are at least at threat level-2. We suggest having at least two different locations inside our software where we do this checking.
2) Sometimes our current geographical location can tell us that the level is 3 or 4. In general it is very difficult to differentiate between level 3 and 4. Therefore, if our software is expected to survive a level 4 attack, then even if we have a level 3 threat, we should treat it as level 4. If on the other hand the maximum level we are trying to protect against is level 3 (not a national security threat), then our reaction to the threat could be economically-based (see actions below).
3) If our device has a heartbeat component and the heartbeat device reports an anomaly. The threat should be level 3 or 4.
4) If our computer is connected to any foreign device [10] (like a foreign type of computer, strange keyboards, printers, etc.). The threat should be level 3 or 4.
5) If we are at a non-friendly location. Minimum threat level is 2. Here the software can be designed to differentiate between non-friendly (i.e. Venezuela), bad-non-friendly (i.e. Iran) and very-bad-non-friendly (i.e. China), and set the threat accordingly.
6) If we are at a friendly location, and there are no signs of tampering, then we can consider at this time the user to be friendly and of no-threat.

B. Actions

General rules of what to do if the secure self-protected software has detected a threat [10]:

- Act accordingly to the defined rules of the self-protected software.
- Notify the secure server of such a threat, including the response given to such a threat.
- Ask the secure server to send back a signal responding to the threat, and either ok or give new response orders. This is done because in all cases, the secure server is the one up-to-date on threats and responses to threats. All individual secure self-protected software will update their response to threats periodically.

On the following actions we are going to suggest a list of possible actions to the given scenario. The actions are:

1) **Minimum:** This is the minimum action that should be taken.
2) **Mild:** An action that recognizes the threat but acts in a manner that the damage to the user is minimal.
3) **Strong**: Act as violently as possible against the user.

Independently of the design decision of location and migration strategies, the actions to take will be:

For a Level-1 casual attacker on any location, we recommend taking all or some of the following actions:

1) Erase all files related to our software located in the user's secure software (Mild).
2) Block any future connections of the user's secure software to our server. Use user's IP and/or user's ID (Minimum).
3) Erase most of the user files (Strong).
4) Send a signal home reporting the issue (Minimum).
5) Display a message to the user saying that a malicious virus has taken control (the idea is to scare the user and mislead him/her on the source of the problem) (Strong).

For a level-2 hacker attack when we are at a friendly location and are guarding a maximum level-2 attack, we recommend taking all or some of the following actions:

1) Same as actions 1-3 of casual attacker.
2) Display a message to the user saying that his/her actions are being reported to the FBI, CIA, Interpol, etc. (Mild).

If on the other hand a hacker attack is detected, and we are guarding against Level-3 or Level-4 attack, or we are at an unfriendly location, we propose the following actions, and that most of the actions be done covertly.

1) Change the software slightly (these actions should be determined at the design stage of the secure software) so it produces results, but the wrong results (Strong).
2) Insert a malicious virus that spreads to all the contacts of this user (Strong).
3) Start a time bomb to damage the local equipment. The due time could be up to a couple of days in order to give time in case the threat changes to a friendly location. If a physical bomb is not possible, then at the due time destroy as much software and information as possible. The destruction of the local equipment should never include human lives (Strong).

VI. Cyber-Ecology

Cyber-Ecology refers to both the scientific analysis and study of the interaction among cyber users and their environment, and the political movement that seeks to protect cyber-space, especially from pollution (i.e. garbage data, viruses, frivolous usage, etc.) [13]-[16]. With this in mind, our model will try to use the Internet at such times at which the load causes less problems for us and for others. In other words, we will use whenever possible the loading and downloading from the server at times when the traffic in the Internet is at low points.

Our software will be monitoring Internet traffic for the specific user and select the best times to do the loading or downloading.

VII. Conclusions

The main contribution of this paper is the explicit use of a web server to protect some of our software that is going to be used as a standalone software by all kinds of users in many countries around the world. We defined an encryption processing algorithm that will increase the security of the software process. We also outline some of the threats and actions that could be taken for different scenarios.

Looking at the state of the cyber world as it is today, we can safely say that the proposed methodology will not work for all

cases of secure software. We have stated that for applications that need a hard response time, this methodology will not work. But it should also be clear that as the speed of the Internet increases more and more, and if in the future there are policies and procedures for which a response time is guaranteed, then this technology becomes a very basic option.

If in the future, the speed of the Internet is not an issue for the use as a software safety methodology, then we have to concentrate our security worries and research in the transmission of the data [17], [18] between the server and the secure software. Also, we have to include in these concerns and research, the security of the web server [2], [19]-[21].

REFERENCES

[1] Intel Corp., "Intel® Data Protection Technology for Transactions ", http://www.intel.com/content/www/us/en/embedded/technology/security/secure-payment-transactions/overview.html, Viewed Jun 2016.
[2] Microsoft," Secure Windows Server", https://technet.microsoft.com/en-us/library/dd548350(v=ws.10).aspx, Viewed June 2016.
[3] Microsoft, "What's New in DHCP", https://technet.microsoft.com/en-us/library/dn765482.aspx, Viewed June 2016.
[4] Hewlett Packard, "HP Advanced Memory Protection technologies", ftp://ftp.hp.com/pub/c-products/servers/options/c00256943.pdf, technology brief, 5th edition, April 2008.
[5] Trend Micro, "Devising a Server Protection Strategy with Trend Micro", http://www.trendmicro.com/cloud-content/us/pdfs/business/white-papers/wp_devise-a-server-protection-strategy.pdf, January 2012.
[6] Yuan E., et al., "A Systematic Survey of Self-Protecting Software Systems", ACM Transactions on Autonomous and Adaptive Systems (TAAS), Volume 8 Issue 4, January 2014, Article No. 17.
[7] Lavasoft, "Potentially Unwanted Program Self-Protection Technologies", http://lavasoft.com/mylavasoft/securitycenter/whitepapers/potentially-unwanted-program-selfprotection-technologies, October 2014.
[8] Lavasoft, "Potentially unwanted programs that use rootkit components, http://lavasoft.com/mylavasoft/securitycenter/whitepapers/pups-with-rootkit, September 2014.
[9] Feiman Joseph, "Runtime Self Protection: A Must Have, Emerging Security Technology", Gartner Group, 24 April 2012.
[10] Gonzalez C. "User Detection in Secure Self-Protected Software", Submitted to ROMJIST Sep 2015.
[11] Denning, Dorothy E., "An Intrusion Detection Model," Proceedings of the Seventh IEEE Symposium on Security and Privacy, May 1986, pages 119-131.
[12] Scarfone, Karen; Mell, Peter. "Guide to Intrusion Detection and Prevention Systems (IDPS)". Computer Security Resource Center (National Institute of Standards and Technology) (800-94) (February 2007).
[13] Jorgensen J., et al., "Cyber Ecology: Looking to Ecology for Insights into Information Assurance", Proceedings of DISCEX 2001, IEEE, 287-296.
[14] Gorman SP., and Malecki EJ.," Fixed and fluid: stability and change in the geography of the Internet", Telecommunications Policy 26 (7), 389-413.
[15] Gupta Ajay and Sekar R.," An Approach for Detecting Self-Propagating Email Using Anomaly Detection", Recent Advances in Intrusion Detection, 2003 – Springer.
[16] Wang Y., et al., "Software Diversity Measurement for Security Evaluation: An Ecological Approach", I. J. Computer Network and Information Security, 2015, 4, 37-43.
[17] Mukherjee B., Heberlein L.T, Levitt K. N.,"Network Intrusion Detection", IEEE Network May 1994.
[18] Creston News Advertiser, "Protect yourself in the online, social network community", 11 Feb 2011. http://www.crestonnewsadvertiser.com/articles/ara/2011/02/11/80449607 08/index.xm
[19] Scarfone Karen, Mell Peter., "Guide to Intrusion Detection and Prevention Systems (IDPS)", Computer Security Resource Center (National Institute of Standards and Technology), February 2007.
[20] Microsoft, "Improving Web Application Security: Threats and Countermeasures", *msdn.microsoft.com*. Viewed June 2016.
[21] *University of Alabama at Birmingham Business Program*. "Information Security: A Growing Need of Businesses and Industries Worldwide". http://businessdegrees.uab.edu/resources/infographics/mis-security-infographic/, Viewed May 2016.